Numerical Methods ITGS219

Lecture 5

Root Finding

Root Finding

- In many problems we are required to determine when a function is zero.
- We now re-introduce the Matlab function ***feval*** which may seem a little like an extra step, but it will enable us to write more general codes. This function has the syntax *feval(f,x_1,...,x_n)*.

```
feval('sin',0.3)
feval('mycode',0.2,0.3)
```

In the first example this gives sin(0.3)

and in the second it returns the value Of mycode(0.2,0.3).

Initial Estimates

- In order to determine a root it is usually essential to have an initial estimate of its value. In some cases you may have more than one root (or none) and you wish to identify which one you are concerned with.
- In general the methods we will consider will require either an initial guess for the root or a bracketing interval containing a root.

Root Finding(Cont.)

We will use the graphical capabilities of Matlab to identify the root (or this bracketing interval).

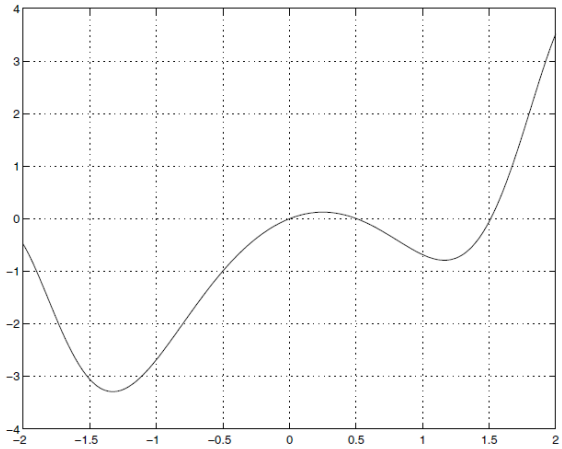
- Firstly, we shall set up a small m-file called *userfn.m*, which will used to specify an example function.

```
function [value] = userfn(x,par1,par2)
value = x-par1*sin(x.^par2);
```

- This gives $f(x) = x - \alpha_1 \sin x^{\alpha 2}$ where α_1 and α_2 are parameters, which the user will specify.

- Now in order to plot the function we select a range and use the plot command. The commands are

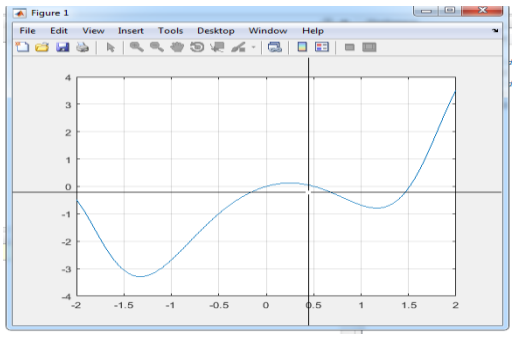
```
x = -2.0:0.01:2.0;
y = feval('userfn',x,2,2);
plot(x,y)
grid on
```



- From this initial figure it is not that clear where the zeros lie. There appears to be one at the origin, which we can see straight away from consideration of the function, $f(x) = x - 2 \sin x^2$.
- By adding the command grid on (either to the programme *testplot.m* or at the command line) we can see two additional zeros (one near 0.5 and another near 1.5)

Root Finding(Cont..)

- In order to investigate further we use the zoom on command. By clicking the left-hand mouse button we can enlarge areas of the figure, and using the right-hand button we can pull back. Typing zoom off disables this feature.
- This can give us a good idea of where the roots lie.
- Notice this command is only working using the original data and if we zoom too close we will be able to see the straight line segments used for the plotting. (401 points).



We can now use another command ***ginput*** to actually return a value rather than just using our eyes. The syntax we will exploit is

```
% Modified testplotm.m
x = -2.0:0.01:2.0;
y = feval('userfn',x,2,2);
disp('Click the mouse near the zero')
disp('and when you have finished press')
disp('the return key')
plot(x,y)
grid on
[xvalues,yvalues] = ginput
[yy,ii] = min(abs(yvalues));
disp(xvalues(ii))
```

- The first few lines are the same as *testplot.m* and the next line uses the command *ginput*. The way *ginput* works is to allow the user to select points in the window (using a cross hair) by clicking one of the mouse buttons. This is terminated by pressing the return button.
- The results are stored in the arrays xvalues and yvalues; these essentially contain the *x and y coordinates of all* the points which the user clicks.
- The next command finds the minimum value of the function and its integer location within the array.
- Finally the value of *x* at this location is displayed. This gave a guess of 0.4977 for the zero near 0.5 and 1.4931 for the zero near 1.5.

Root Finding(Cont..)

Editor - C:\Program Files\MATLAB\TestCommandsLec01\userfn.m

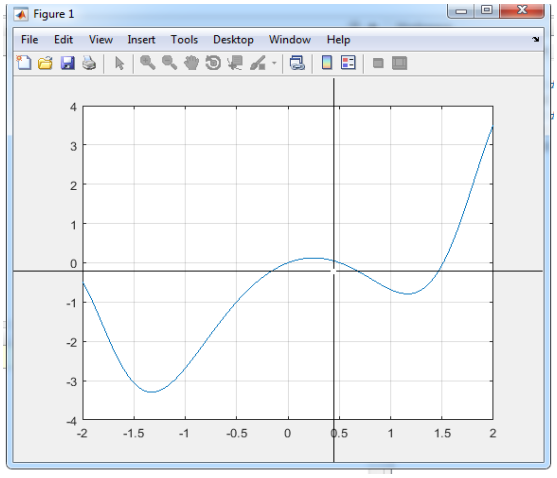
first.m | userfn.m

```
1   function [value] = userfn(x,par1,par2)
2 -   value = x-par1*sin(x.^par2);
3 -   end
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> % Modified testplotm.m
x = -2.0:0.01:2.0;
y = feval('userfn',x,2,2);
disp('Click the mouse near the zero')
disp('and when you have finished press')
disp('the return key')
plot(x,y)
grid on
[xvalues,yvalues] = ginput
[yy,ii] = min(abs(yvalues));
disp(xvalues(ii))
Click the mouse near the zero
and when you have finished press
the return key
```



Root Finding(Cont…)

Example 4.1 Determine initial estimates for the zeros of the function ***f(x)*** =*x sin x* − √ *x between 0 and 10.*

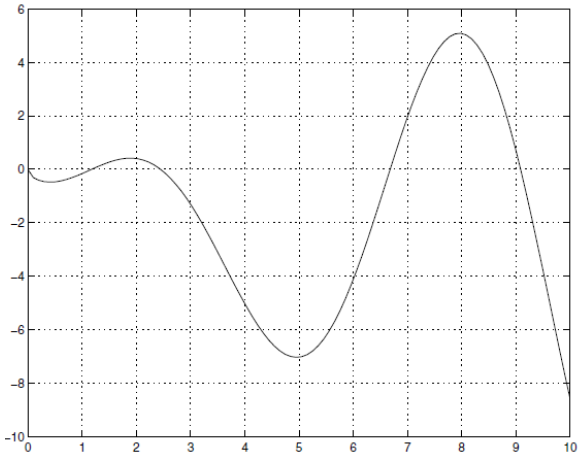
- Firstly we construct the Matlab code:

```
function [value] = userf1(x)
value = x.*sin(x)-sqrt(x);
```

- and now run the commands

```
>> x = linspace(0,10);
>> y = feval('userf1',x);
>>plot(x,y)
>>grid on
```

which yield the next figure =>



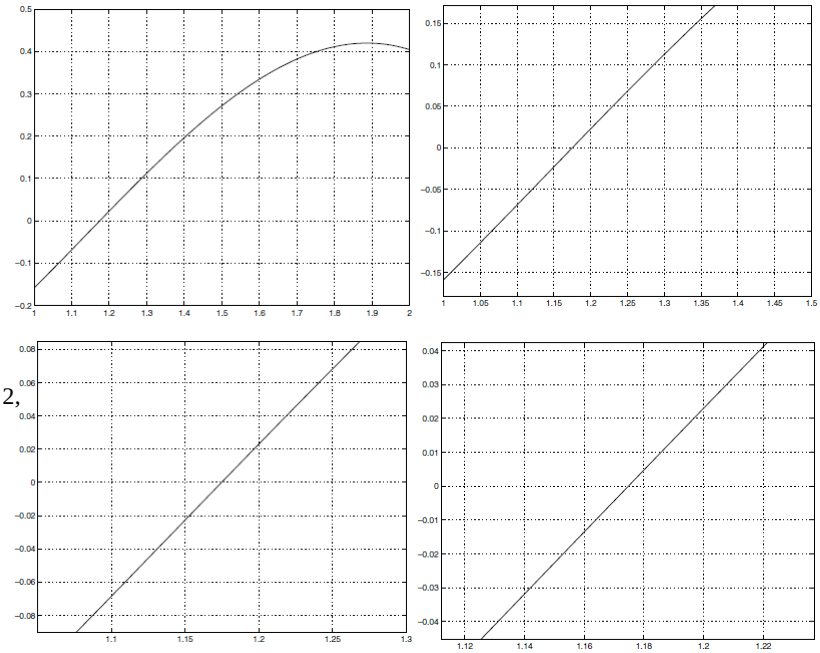
- We can see that there are zeros at 0 and near 1.2, 2.5, 6.7 and 9.3.
- We could use the above data set but it would seem sensible to consider each point separately.
- For this purpose we use the code on the next slide:

Root Finding(Cont….)

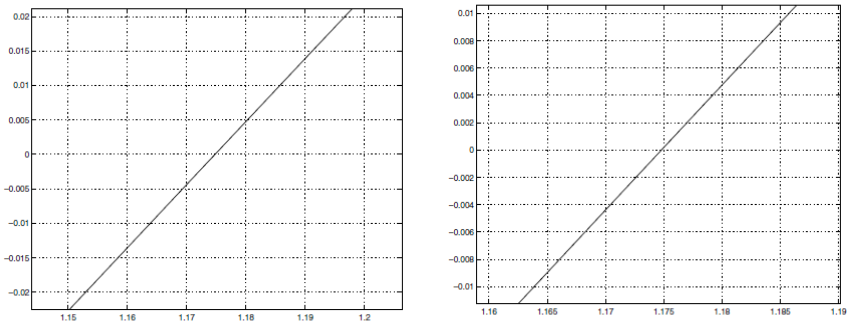
- For this purpose we use the code:

```
a = input('Start of interval ');
b = input('End of interval ');
x = linspace(a,b);
y = feval('userf1',x);
clf
plot(x,y)
grid on
zoom on
```

- Running this code with the inputs 1 and 2, we obtain:



Root Finding(Cont…..)



- after successive clicks of the left mouse button. This allows us to obtain a better estimate of the root, namely 1.175.
- You should try this interval and repeat the exercise for the other roots.
- In reality you should plot the function and then use a combination of the grid and zoom commands to obtain estimates for the root or intervals which contain them. This also helps to ensure that the answers you obtain are sensible.

Root Finding (Fixed Point Iteration)

- Most of the techniques we will discuss are iterative in nature and the first one is called the **fixed point iteration** scheme.
- Instead of looking for a zero of the function *f(x), it determines a fixed point of an equivalent equation.*
- The equation is rewritten in the form *x = g(x), so when x is substituted in the* function *g(x) it returns the value x,* hence the nomenclature fixed point.
- The conversion of the equation *f(x) = 0 into one of the form x = g(x) is not* always straightforward and definitely can be done in many ways. For instance the previous example, for which *f(x) = x − 2 sin x² could be rewritten as*
- *x = 2sin x² (where g(x) = 2sin x²) or* $x = \sqrt{\sin^{-1}\frac{x}{2}}$.
- For instance consider the quadratic *f(x) = x² + 2x − 3, which could* be manipulated to give *x = (3 − x²)/2 or x =√(3 − 2x).*
- With these two forms neither of them seems to have any advantages over the other.
- Further the equation could be written as *x = 3/(x+2) and so on.*
- *Before we try to resolve* this issue let us say how this is then implemented as a numerical scheme.
- We rewrite the equation as $x_{n+1} = g(x_n) \qquad n = 0, 1, \cdots,$

Root Finding (Fixed Point Iteration Cont.)

- which is a recursion formula. It starts with an initial guess, namely x_0 *(which* may be determined graphically or by another means).
- Just about the simplest code for this purpose would be

```
x0 = 1;
for j=1:10
   x0 = g(x0);
end
```

- where we have a routine *g.m* which defines the function *g(x) and* $x_0 = 1$ *is a* suitable initial guess.
- This runs through the iterative process ten times. This kind of code presupposes that it is going to work and will **converge** in ten steps (or an appropriate number).
- what we mean by convergence?
- We could work out the value of the function *f(x) at the current iterate as a check. We* would require this to be less than a certain tolerance (the accuracy to which we would expect to know the answer).
- Notice that this is slightly different to knowing the root to within a certain tolerance.
- Alternatively in this case the code may be deemed to be successful when the difference between x_{n+1} *and* x_n *is less than a certain tolerance. In this case* we have $x_{n+1} \approx x_n \Rightarrow x_n \approx g(x_n) \Rightarrow f(x_n) \approx 0$.

Root Finding (Fixed Point Iteration Cont..)

This tolerance reflects how well we want to know the answer
and the parameter **maxits** is how many times we are prepared to perform the iterations.
This is to eliminate problems with cases which don't converge and hence cause infinite loops

```
% fixed.m
function [answer,iflag] = fixed(g,xinit)
global tolerance maxits
iflag = 0;
iterations = 0 ;
xnext = feval(g,xinit);
while (iterations<maxits) & abs(xnext-xinit)>tolerance
    iterations = iterations + 1;
    xinit = xnext;
    xnext = feval(g, xinit);
end
if iterations == maxits
  iflag = -1;
  answer = NaN;
else
  iflag = iterations;
  answer = xnext;
end
```

```
% eqn.m
function [g] = eqn(x)
g = 2*sin(x.^2);
```

main function

```
% mfixed.m
global tolerance maxits
tolerance = 1e-4;
maxits = 30;
[root,iflag] = fixed('eqn',0.2);
switch iflag
      case -1
         disp('Root finding failed')
      otherwise
        disp([' Root = ' num2str(root) ...
         ' found in ' num2str(iflag) ' iterations'])
end
```

Root Finding (Fixed Point Iteration Cont..)

- We have exploited the new command global which allows variables to be used by any routine which contains a global statement referring to the same variables, or a subset of them.
- The code above is run and gives the result of $\approx 1 \times 10^{-14}$ *after only four* iterations.
- In order to identify the other roots (near 1/2 and 3/2) we must use guesses close to these values.
- Starting around 0.5 (for instance guesses of 0.4 and 0.6 lead to the code finding the root at zero again). Starting close to the other root has a variety of outcomes: starting below it produces the zero root, whereas above leads to the iterations diverging.
- The reason for this is linked to the derivative of $g(x)$. *In fact the errors are multiplied by* $|g(x)|$ *at each* iteration and if this value is **greater than one**, **the method will fail**.

○ In this case we recall that $g(x) = 2\sin x^2$ so that:

- $g(x) = 4x \cos x^2$ *which when* $x = 12$ equals 1.9378 and when $x = 32$ gives -3.7690.
- Near the origin the derivative is very small and consequently the technique works well.

Root Finding (Fixed Point Iteration Cont…)

Fixed point : A point, say, ***s*** is called a fixed point if it satisfies the equation **x = g(x)**.

Fixed point Iteration : The transcendental equation **f(x) = 0** can be converted algebraically into the form **x=g(x)** and then using the iterative scheme with the recursive relation $x_{i+1} = g(x_i)$; $i = 0, 1, 2, \ldots$

- with some initial guess x_0 is called the *fixed point iterative* scheme.

Algorithm - Fixed Point Iteration Scheme

- Given an equation $f(x) = 0$
- Convert $f(x) = 0$ into the form $x = g(x)$
- Let the initial guess be x_0
- Do

  $x_{i+1} = g(x_i)$

  while (none of the convergence criterion C_1 or C_2 is met)
- C_1. Fixed a priori the **total number of iterations N** .
- C_2. By testing the condition | x_{i+1} - $g(x_i)$ | < **tolerance limit** (where ***i*** is the iteration number), say epsilon, **fixed a priori**.

Condition for Convergence:

- If $g(x)$ and $g'(x)$ are continuous on an interval J about their root s of the equation $x = g(x)$,
- and if $|g'(x)|<1$ for all x in the interval J then the fixed point iterative process $x_{i+1}=g(x_i)$, $i = 0, 1, 2, \ldots$,

  will converge to the root $x = s$ for any initial approximation x_0 belongs to the interval J .

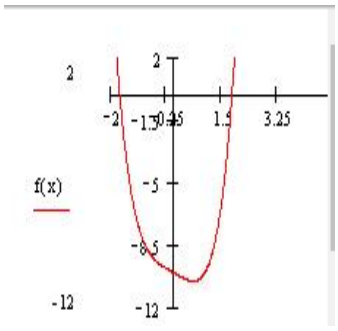
Root Finding (Fixed Point Iteration Cont…)

Example: Find a root of $x^4-x-10 = 0$ by using fixed point iterative scheme

- Consider $g_1(x) = x = 10 / (x^3-1)$ and the fixed point iterative scheme $x_{i+1} = g(x_i)$ so that $x_{i+1}=10 / (x_i^3 -1)$, for $i = 0, 1, 2, \ldots$ And let the initial guess $x_0 = 2.0$

i	0	1	2	3	4	5	6	7	8
x_i	2	1.429	5.214	0.071	-10.004	-9.978E-3	-10	-9.99E-3	-10

So, the iterative process with g_1 gone into an *infinite loop without converging*.



- Consider another function $g_2(x) = (x + 10)^{1/4}$ and the fixed point iterative scheme $x_{i+1} = (x_i + 10)^{1/4}$; $i = 0, 1, 2, \ldots$ And let the initial guess x_0 *be* ***1.0, 2.0*** *and* ***4.0***

i	0	1	2	3	4	5	6
x_i	1.0	1.82116	1.85424	1.85553	1.85558	1.85558	
x_i	2.0	1.861	1.8558	1.85559	1.85558	1.85558	
x_i	4.0	1.93434	1.85866	1.8557	1.85559	1.85558	1.85558

That is for g_2 the iterative process is converging to **1.85558 with** any initial guess.

Root Finding (Fixed Point Iteration Cont…)

- Consider $g_3(x) = (x+10)^{1/2}/x$ and the fixed point iterative scheme

  $x_{i+1} = (x_i+10)^{1/2}/x_i$, $i = 0, 1, 2, \ldots$ let the initial guess $x_0 = 1.8$,

i	0	1	2	3	4	5	6	…	98
x_i	1.8	1.9084	1.80825	1.90035	1.81529	1,89355	1.82129	…	1.8555

  That is for g_3 with any initial guess the iterative process is converging but very slowly

- The Geometric interpretation of convergence with g1, g2 and g3

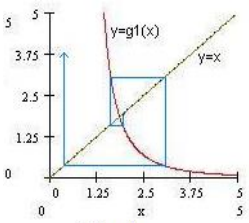


Fig g1

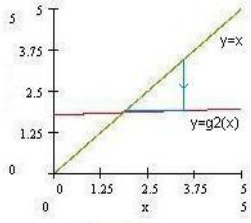


Fig g2

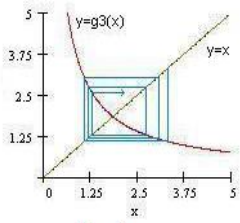


Fig g3

The graphs Fig g1, Fig g2 and Fig g3 demonstrates the Fixed point Iterative Scheme with g1, g2 and g3 respectively for some initial approximations.

It's clear from the -Fig g1, the iterative process diverges (does not converge) for any initial approximation.

-Fig g2, the iterative process converges very quickly to the root which is the intersection point of y = x and y = $g_2(x)$ as shown in the figure.

-Fig g3, the iterative process converges but very slowly.

Fixed Point Iteration (Condition for Convergence)

HomeWork

1 Find a root of cos(x) - x * exp(x) = 0
2 Find a root of x^4-x-10 = 0
3 Find a root of x-exp(-x) = 0
4 Find a root of exp(-x) * (x^2-5x+2) + 1= 0
5 Find a root of x-sin(x)-(1/2)= 0
6 Find a root of exp(-x) = 3log(x)

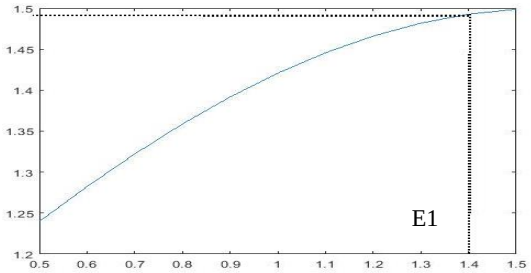
Fixed Point Iteration (Conditions of Convergence)

Condition for Convergence:

E1: $x_{n+1} = 1 + 0.5 \sin x_n$

E2: $x_{n+1} = 3 + 2 \sin x_n$

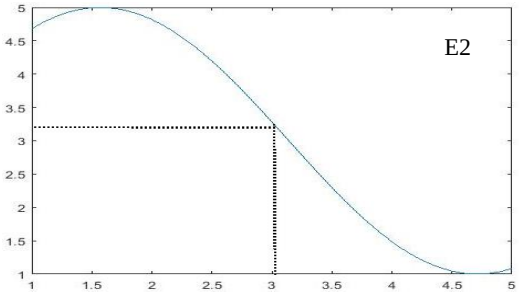
for $n = 0, 1, 2, \ldots$



The solutions are

E1: $\alpha = 1.49870113351785$

E2: $\alpha = 3.09438341304928$



Fixed Point Iteration (Condition for Convergence)

Condition for Convergence :

In the earlier example.

- First consider E1: $x = 1 + 0.5 \sin x$ for $i=0$
- Here $g(x) = 1 + 0.5 \sin x$ We can take [a, b] with any $a \leq 0.5$ and $b \geq 1.5$.
- Note that $g'(x) = 0.5 \cos x$; $|g'(x)| \leq 1/2$
- Therefore, we can apply the theorem and conclude that the fixed point iteration
- $x_{n+1} = 1 + 0.5 \sin x_n$ will **converge** for E1.

Then we consider the second equation

- E2: $x = 3 + 2 \sin x$
- Here $g(x) = 3 + 2 \sin x$
- Note that $g(x) = 3 + 2 \sin x$; $g'(x) = 2 \cos x$ and $g'(\alpha) = 2 \cos (3.09438341304928) = -1.998$
- Therefore the fixed point iteration $x_{n+1} = 3 + 2 \sin x_n$ will **diverge** for E2.

You are welcome for

Any Question?